Modeling the Object-Oriented Space Through Validated Measures

Ralph D. Neal
West Virginia University
NASA/WVU Software Research Laboratory
100 University Drive
Fairmont, WV 26554
304-367-8355
rneal@research.ivv.nasa.gov

Abstract—In order to truly understand software and the software development process, software measurement must be better understood. A beginning step toward a better understanding of software measurement is the categorization of the measurements by some meaningful taxonomy. The most meaningful taxonomy would capture the basic nature of the object-oriented (O-O) space. The interesting characteristics of object-oriented software offer a starting point for such a categorization of measures. A taxonomy has been developed based upon fourteen characteristics of object-oriented software gathered from the literature. This taxonomy allows us to easily see gaps and redundancies in the O-O measures. The taxonomy also clearly differentiates among taxa so that there is no ambiguity as to the taxon to which a measure belongs. The taxonomy has been populated with thirty-two measures that have been validated in the narrow sense of Fenton [9] using measurement theory with Zuse's [30] augmentation.[1]

TABLE OF CONTENTS

1. INTRODUCTION

Software development historically has been the arena of the artist. Artistically developed code often resulted in arcane algorithms or spaghetti code that was unintelligible to those who had to perform maintenance. Initially only very primitive measures such as lines of code (LOC) and development time per stage of the development life cycle were collected. Projects often exceeded estimated time and budget. In the pursuit of greater productivity, software development evolved into software engineering. Part of the software engineering concept is the idea that the product should be controllable. Control of a process or product requires that the process or product is measurable; therefore, control of software requires software measures [2].

Software and software development are extremely complex. We should not expect to measure something so complex with one, two, or even a dozen measures. Measures have to be developed to allow us to view software

[1] Funded in part by NASA Cooperative Agreement NCCW-0040

from many perspectives. Many object-oriented (O-O) metrics have been proposed in the literature, e.g., [1], [6], [7], [16], and [17]. To better understand the contribution of these metrics, it is necessary to categorize them so that we can better understand the dimensions of O-O software. No one has yet organized these metrics in any way that models the O-O space. Until we understand the many dimensions of O-O software, we cannot truly understand the product. It does no good to measure the process if the product is not measured. Being the best at producing an inferior product does not define a quality process. To facilitate understanding of the product, this paper proposes a taxonomy that not only allows us to classify measures but also helps us model the object-oriented space.

Definition of Terms and Notation

Complexity, inter-structural	Complexity introduced into the structure of a class by the interaction with other classes.
Complexity, intra-structural	Complexity introduced into the structure of a class by the interaction of methods within the class.
Complexity, psychological	Non-structural concepts which make understanding of the entity being measured more difficult.
Cohesion	The extent to which a class is self-contained.
Coupling	The extent to which a class utilizes attributes outside itself.
Entity	An object or an event, e.g., a developed program or the development process.
Measure (n.)	A numeric representation that has been validated to measure some dimension of some entity -- in our case, software.
Measurement	The process of empirical, objective assignment of numbers to the properties of objects and events in the real world in such a way as to describe them [Zuse, 1991].
Measurement level	Defines the scope of the measurement by naming the thing being measured, i.e., variable, method, class, program, or system.
Metric	A numeric representation (not necessarily validated) that purports to measure some dimension of software.
Program	A collection of classes to perform a specific function, e.g., a payroll deduction calculation program.
Ratio scale	Represents ratios of a property, i.e., ratio scales allow statements such as "*a* is twice as complex as *b*" (iff (*a*)=2 (*b*)); assigned to measures that fulfill the extensive structure.
Ratio’ scale	A weak ratio scale; assigned to measures that fulfill the concatenation rules of the set theory union operation.
Software dimension	An interesting characteristic of the software. Dimensions are the characteristics which we are interested in measuring.
System	A coordinated collection of programs to carry out a specific procedure, e.g., a payroll system.
Taxa	Plural of taxon.
Taxon	A taxonomic category or unit.
Taxonomy	An arrangement of items (measures) onto natural,

	related groups based on factors common to each.
E	observed relational system (Empirical Relational System: A=(A,R1,...,Rn))
ℙ	some property of software which we wish to measure.
R	numerical relational system (Formal Relational System: B=(B,S1,...,Sn).).
➢	signifies is larger than (or is preferred to).
~	signifies is equivalent to (or is indifferent to).
$\rightarrow$	onto
✡	necessarily leads to.
◎	binary operation in the empirical relational system, usually designated concatenation.

2. THE OBJECT-ORIENTED SPACE

Authors have not been in agreement about the characteristics that identify the object-oriented approach. Henderson-Sellers [12] listed information hiding, encapsulation, objects, classification, classes, abstraction, inheritance, polymorphism, dynamic binding, persistence, and composition as having been chosen by at least one author as a defining aspect of object-orientation. Rumbaugh, et al. [20] added identity, Smith [22] added single type and Sully [23] added the unit building block to this list of defining aspects. These aspects of object-orientation are not disjoint. In fact there is much overlapping of aspects as different authors grouped sub-aspects differently and created their own individual groupings, each with a unique aspect name. It should be obvious from the preceding list that there are many dimensions to O-O software. It should also be noted that this list may not be exhaustive.

The Tegarden, et al., [24] model of object-oriented systems complexity measurement defines object-oriented systems as looking different from different viewpoints. This model defines four levels of software strata. [18] adds a fifth level of strata. Building on this model, the object-oriented space can be represented as a matrix that partitions the space into levels of granularity. The software levels that a software developer might want to measure (in order of granularity) are: variables, methods, classes, programs, and systems. Each level of granularity exhibits characteristics that contribute to the character of that level. Designating the five levels of granularity as columns and fourteen dimensions of O-O software that have been gleaned from the literature as rows, The Object-Oriented Space matrix (see Table 1 for axes headings) is proposed. This model forces a reasonable consensus upon measurers.

Table 1: The Object-Oriented Space

Software Dimensions (column headings)	Levels of granularity (row headings)
Clarity	Variable
Cohesion	Method
Coupling	Class
Complexity, inter-structural	Program
Complexity, intra-structural	System
Complexity, psychological	
Design	
Encapsulation	
Inheritance	
Information hiding	
Polymorphism	
Reuse	
Size	

Specialization	

In order to measure object-oriented software the measurer will need to be aware of the characteristics of O-O software and of the different levels of granularity inherent in the O-O paradigm. This model parses the object-oriented space into understandable, unambiguous segments and allows the measurer to reason about the object-oriented space in a meaningful way.

3. SOFTWARE MEASUREMENT

Many researchers have asserted the importance of software measurement. Vollman [26] wrote of the importance of software measurement to society while Grady and Caswell [11] and Chidamber and Kemerer [7] described its importance to management.

Measurement is the process whereby numbers or symbols are assigned to dimensions of entities in such a manner as to describe the dimension in a meaningful way. An entity may be a thing or an event, e.g., a person, a play, a developed program or the development process. A dimension is a trait of the entity, such as the height of a person, the cost of a play, or the length of the development process. Obviously, the entity and the dimension to be measured must be specified in advance. Measurements cannot be taken and then applied to just any dimensions. Unfortunately this is exactly what the software development community has been doing [10], e.g., lines-of-code, being a valid measurement of size, has been used to "measure" the complexity of programs [28].

An intuitive and empirical assessment of the entities and dimensions must be preserved by the measurement (the assignment of numbers and symbols). For example, when measuring the height of two people the taller person should be assigned a larger number than the shorter person. Notice that the unit of measure, (feet, inches, meters) has no effect on this rule. Likewise, when measuring software complexity, the more complex program should be assigned a larger number than the less complex program. This is discussed in depth in the section on Measurement Theory.

Because people observe things differently (and often intuitively feel differently about things), a model is usually defined for the entities and dimensions to be measured. The model requires everyone to look at the subject from the same viewpoint. Fenton [10] uses the example of human height. Should posture be taken into consideration when measuring human height? Should shoes be allowed? Should the measurement be made to the top of the head or the top of the hair? The model forces a reasonable consensus upon the measurers. This idea is applied to software measurement in the section on The Object-Oriented Space.

There are two types of measurement: *direct measurement* of a dimension requires only that dimension; *indirect measurement* of a dimension requires that one or more other dimensions be measured. Because the dimensions of greatest interest, e.g., quality and reliability, are often external to the entity being measured, and therefore very hard to measure directly, indirect measurement usually achieves more useful results [10]; [15]. That is, internal dimensions e.g., verbal skills, are measured, to assess external dimensions, e.g., intelligence quotient (IQ). Or in the case of software, the number of known defects are counted to assess quality.

4. THE IMPORTANCE OF VALIDATION

Fenton [10] argued that much of the software measurement work published to date is scientifically flawed. This is not a revelation. Software metrics usually have been taken at face value. Because many people believe that any quantification is better than no quantification at all, just counting the lines of code (for example) was enough to give management the feeling of doing something to try to gain control of the software development process. After obtaining the quantification, management had to try to decide just what was described and how the development process was influenced. Fenton [9] stated that it is often the case that the general lack of validation of software metrics is the reason that managers do not know what to do with the numbers with which they are presented.

Fenton is not the only author who has observed this lack of scientific precision. Baker, et al., [2] said as much when they wrote that research in software metrics often is suspect because of a lack of theoretical rigor. Li and Henry [16] argued that validation is necessary for the effective use of software metrics. Schneidewind [21] stated that metrics must be validated to determine whether they measure what it is they are alleged to measure. Weyuker [27] stated that existing and proposed software measures must be subjected to an explicit and formal analysis to define the soundness of their properties.

Fenton [9] described two meanings of validation. Validation in the narrow sense is the rigorous process of ensuring that the measure properly represents the intended dimension of the software, i.e., verify that the measure is theoretically sound. Validation in the wide sense is the authentication of a prediction system using verified measures of the software. Accurate prediction relies on careful measurement of the predictive dimensions. A model which accurately measures the dimensions is necessary but not sufficient for building an accurate prediction system. The model, along with procedures for determining the parameters to feed the model, and procedures to elucidate the results all are necessary to build an accurate prediction system [9].

In the past, validation in the wide sense has been conducted without first carrying out validation in the narrow sense. Validation in the narrow sense is a necessary step before measures can be used to predict such managerial concerns as cost, reliability, and productivity.

> "Very few metrics have been proposed to measure object oriented systems, and the proposed ones have not been validated." [16]

Since Li and Henry's statement, there has been an explosion of object-oriented software metrics. The recent flood of object-oriented software metrics (Chen and Lu [6]; Li and Henry [16]; Chidamber and Kemerer [7]; and Lorenz and Kidd [17]) has hit the scene with limited validation beyond regression analysis of observed behavior. Chidamber and Kemerer dedicated a sub-section to measurement theory within the section devoted to the research problem. They explained empirical relation systems, formal relational systems, mapping from the empirical system to the relational system, and the properties of the weak order. However, they made no reference to measurement theory in the section on metrics evaluation criteria. They made no attempt to assign a scale to their metrics nor to evaluate them vis-a-vis the representation and uniqueness theorems of measurement theory.

5. MEASUREMENT THEORY

So we have seen, many metrics have been used without the benefit of any theoretical validation. Fenton [9] writes that measures must be validated in the narrow sense using measurement theory. Fenton's narrow validation is required to establish the scale of the measure in order to know which statistics can be legitimately applied. Briand, et al. [3] write that measurement theory, while valid for the structured paradigm, does not migrate to the object-oriented paradigm. Zuse [30] writes that the *Dempster-Shafer Function of Belief* allows us to substitute set theory for the intensive structure of measurement theory to validate measures in the narrow sense of Fenton.

The task of measurement theory is to categorize and describe the types of measurement. There are two fundamental problems in measurement theory; the first is the *representation problem*. The representation problem is to find sufficient conditions for the existence of a mapping from an observed system to a given mathematical system. More formally, given a particular empirical relational system E and a numerical relational system R, find sufficient conditions for the existence of a mapping from E into R. The sufficient conditions, referred to as representation axioms, specify conditions under which measurement can be performed.

Fenton [9] used human height to explain the representation problem. Suppose three people are present. It may be observed that Tom is the tallest of the three, Dick is the shortest of the three, and Harry is taller than Dick and shorter than Tom. Thus the *taller than* relationship among the three people has been empirically established. Any measurement taken of the height of these three people must result in numbers or symbols that preserve this relationship. If it is further observed that Tom is *much taller than* Dick, then this relationship must also be preserved by any measurement taken. That is, the numbers or symbols used to represent the heights of Tom and Dick must convey to the observer the fact that Tom is indeed *much* taller than Dick. If it is further observed that Dick towers over Tom when seated on Harry's shoulders, then another relationship has been established which must also be preserved by any measurement taken. This relationship might be represented in the real number system by (.7Dick + .8Harry > Tom). Any numbers that resulted from measuring the height of Tom, Dick, and Harry would have to satisfy the observation represented by our formula. Thus the measurement represents our empirical findings.

Another aspect of the representation problem is pointed out by Weyuker [27]. How unique is the result of the measurement? A measurement system must provide results that enable us to distinguish one class of entity from another class of entity. If a measurement groups all entities into only two or three classes, two entities that are clearly different may end up in the same class, i.e., it may be impossible to discriminate between two entities that should be in different classes. Using Tom, Dick, and Harry again, it is easy to see that measuring their height in miles is less representational than measuring their height in inches, i.e., measured to the closest mile, they are all the same height.

The other fundamental problem of measurement theory is the *uniqueness problem*. Uniqueness theorems define the properties and valid processes of different measurement systems and tell us what type of scale results from the measurement system. Additionally, uniqueness theorems contribute to a theory of scales. According to this theory of scales, the scale used dictates the meaningfulness of statements made about measures based on the scale. [14; 19]

Let us consider two statements: 1) This rock weighs twice as much as that rock; 2) This rock is twice as hot as that rock. The first statement seems to make sense but the second statement may not. The ratio of weights is the same regardless of the unit of measurement while the ratio of temperature depends on the unit of measurement. Weight is a ratio scale, therefore, regardless of whether the weights of the rocks are measured in grams or ounces the ratio of the two is a constant. Fahrenheit and Celsius temperatures are interval scales, i.e., they exhibit uniform distance between integer points but have no natural origin. Because Fahrenheit and Celsius are interval scales, the ratio of the temperatures of the rocks measured on the Fahrenheit scale is different from the ratio when the temperatures are measured on the Celsius scale. A statement involving numerical scales is meaningful if the truth of the statement is maintained when the scale involved is replaced by another (admissible) scale.

The Empirical/Formal Relational System

A relational system is a way of relating one entity (or one event) of a set to another entity (or event) of the same set. In the physical sciences the relations take the form longer than, heavier than, of equal volume, etc. In the social sciences (and thus in software measurement) the relations take the form is preferred to, is not preferred to, is at least as good as.

> Definition 1: The <u>ordinal relational system</u> is an ordered tuple (A,R1,...,Rn) where A is a nonempty set of entities and the Ri, i=1,...,n are k-ary relations on A. [28]

The Empirical Relational System is E = (A,R1,...,Rn) where A is a non-empty set of dimensions that are to be measured and the Ri are k-ary empirical relations on A as described above.

The Formal Relational System is R = (B,S1,...,Sn) where B is a non-empty set of formal entities (for example, the real numbers) and the Si are k-ary relations on B such as "equal" or "greater than."

Measurement is a mapping M: A→B such that M preserves the relations in A, i.e., let B be the real numbers, then, if the observed entity *a1* is larger than (or preferred to) observed entity *a2*, then the formal entity *b1* must be greater than formal entity *b2*. Let ➢ denote is larger than (or is preferred to) then M is a valid mapping from A to B iff *a1* ➢ *a2* ✡ *b1* > *b2*.

The relational systems A and B along with the mapping M are sufficient to measure entities on the ordinal scale. If it is enough to know that program module 1 is preferred to program module 2 based on some measurement, then no further structure is needed. However, modules often are combined to create a composite entity which is different from its parts. In order to measure composite entities one must either recalculate the empirical value of the composite entity or combine the empirical values of the parts in some meaningful way.

The Extensive Structure

The extensive structure is an expansion of the ordinal relation system to include binary processes on the entities of the set. The binary process in the empirical relational system usually is designated concatenation, denoted by ◎. The usual manifestation of the binary process in the formal relational system is addition (+) although multiplication may be the proper process under some circumstances.

> Definition 2: The <u>extensive relational system</u> is an ordered tuple (A,R1,...,Rn, ◎1,...,◎m) where A is a nonempty set of entities, the Ri,

i=1,...,n are k-ary relations on A and the ⊚j, j=1,...,m are closed binary operations. [28]

The extensive structure is required to measure entities on the interval or ratio scales. Let b1, b2, b3, b4, be the formal measures associated with a1, a2, a3, a4. Under the extensive structure, M is a homomorphism from A to B iff *a1⊚a2 ➢ a3⊚a4* ✡ *b1+b2 > b3+b4*, i.e., the observed relationship of the concatenated entities must be preserved by the mapping to the formal relationship. Note that addition is assumed to be the concatenation operator.

Criticism of the Extensive Structure

Recent work has questioned the applicability of the extensive structure to object-oriented measures [3], [4], [5], [29], and [30]. Particularly important is the question: must the measurement of an entity formed by the concatenation of two modules equal the sum of the measurements of the independent modules before concatenation, i.e., let b1, b2, be the formal measures associated with a1, a2; is it necessary that *a1⊚a2* equates to *b1+b2*?

The result of combining two classes (C_1, C_2) is a single class (C_3) which combines all of the properties of the two initial classes. There are four ways that C_3 might be formed [13]:

1) C_3 contains C_1 and C_2 and the names of the properties which appear in both have been differentiated to avoid ambiguities. In this instance, the extensive structure holds;

2) C_3 contains both C_1 and C_2 but the instance variables which appear in both are hidden in one and the methods which appear in both are overloaded in that same one. All properties of both classes are present, therefore, the extensive structure holds;

3) C_3 contains C_1 and C_2 as subobjects and does not present their properties to the outside world. Again, all properties are present, and yet again, the extensive structure holds;

4) C_3 is created by merging. Let C_1 be a class with the properties (methods and instance variables) *a, b, c* and C_2 be a class with properties *a, d, e*. Let $C_3 = C_1$ ⊚ C_2, then the properties of C_3 are *a, b, c, d, e*. This is the definition of the *union* operation in set theory. Object-oriented classes then may be viewed as sets with the methods and instance variables of the class being the elements of the set [30].

Concatenation of classes when C_1 and C_2 are merged should follow the rules of set theory. Whenever set theory is used, instead of the extensive structure, to test for a scale above the level of the ordinal scale [30] we will call this the ratio' scale. This is a weaker test than the extensive structure test. The extensive structure dominates the set theory union structure. That is, the set theory union structure always holds if the extensive structure holds, and the set theory union structure may hold when the extensive structure does not.

The question really is: if a measure is ordinal but fails the extensive structure, is the measure strictly ordinal or would much valuable data be lost by not considering the measure as a higher order scale [25 as cited by 3]? Parametric statistics have been shown to be more robust in the face of scale , i.e., more accurate when the wrong scale has been assumed, than nonparametric statistics [4]. Therefore, there are two reasons to extend the scale of a measure to a higher level. The measure may be more powerful than the ordinal scale will reflect and the parametric statistics that we wish to use to take advantage of this power are forgiving of miscategorization of scale.

6. THE MEASURES

The measures taken from [6], [7], [16], [17], and [18] have been validated to be ratio (or ratio') scales. The metrics from Tegarden, et al., have been taken from their paper without validation. They have been included to show that work is being done at the variable and method levels.

This is in no way all of the metrics offered by Tegarden, et al. None of their proposed metrics have been included for cells for which validated measures already exist. Also, some of their metrics were merely observations, e.g., "The complexity measured by the fan-in and fan-out measures increase intervariable complexity and the variable polymorphism measure decreases intervariable complexity." This approach does not convey enough information to allow us to place the variables in order by complexity, i.e., this approach does not lead to an ordinal scale.

Validated Measures

(AIM) Average number of instance methods per class [17]
(AIV) Average number of instance variables [17]
(AMS) Average method size [17]
(CRE) Number of times a class is reused [17]
(CLM) Average number of comment lines per method [17]
(DAC) Density of abstract classes [18]
(DCBO) Degree of coupling between classes [18]
(DCWO) Degree of coupling within classes [18]
(DMC) Density of methodological cohesiveness [18]
(FFU) Use of friend functions [17]
(FOC) Percentage of function-oriented code [17]
(IMC) Intraclass method calls [18]
(LOC) Lines of code [17]: Number of statements (NOS) [17]: Number of semicolons in a class (SIZE1) [16]
(MAA) Messages and arguments [18]
(MPC) Message-passing coupling [16]
(NAC) Number of abstract classes [17]
(NCM) Number of class methods in a class [17]
(NIM) Number of instance methods in a class [17]
(NIV) Number of instance variables in a class [17]
(NMA) Number of methods added by a subclass [17]
(NOM) Number of local methods [16]
(PIM) Number of public instance methods in a class [17]
(PMI) Potential methods inherited [18]
(PMIS) Proportion of methods inherited by a subclass [18]
(POM) Proportion of overriding methods in a subclass [18]
(PRC) Number of problem reports per class or contract [17]
(PrIM) Number of private instance methods [18]
(RFC) Response for a class [7]
(RUS) Reuse [6]
(SML) Strings of message-links [18]
(UCGU) Unnecessary coupling through global usage [18]
(WMC) Weighted methods per class [7]

Metrics Not Yet Validated

(I/Ov) Invoked object variables [24]
(mfd) Method fan down [24]
(mfi) Method fan in [24]
(mfo) Method fan out [24]
(mp) Method polymorphism [24]
(vfd) Variable fan down [24]
(vfi) Variable fan in [24]
(vfo) Variable fan out [24]
(vp) Variable polymorphism [24]

7. THE TAXONOMY

As has been stated, a beginning step toward understanding software measurement is the categorization of the measurements by some meaningful taxonomy. Archer and Stinson [1] propose a taxonomy that places a metric in one (or more) of five taxa, viz., system, coupling and uses, inheritance, class, and method. It is unclear where a measure of say coupling among methods, as in [24], would be classified in this taxonomy. The coarseness of this taxonomy also causes metrics for different software artifacts to be grouped together, e.g., if all coupling metrics are classified as "coupling and uses" metrics, then measures of classes would be lumped together with measures of methods and measures of variables, again as in [24]. A useful taxonomy clearly should differentiate among taxa so that there is no ambiguity as to the taxon to which a measure belongs. If we are to learn about the object-oriented space, it must be possible for diversified measurers to reach the same conclusions given the same data. A taxonomy should at least allow each measurer the ability to start reasoning from the same sensibility.

The Object-Oriented Space matrix offers a starting point for such a categorization of measures. By filling in the cells of the Object-Oriented Space matrix with the measures from section 6, the matrix becomes the Object-Oriented Measures Taxonomy (see Table 2). This taxonomy includes all of the known, interesting characteristics of software and clearly defines where any measure fits among the taxa without worry of overlap or ambiguity. If a measure cannot be placed easily into one and only one taxon, the measure may not be well understood. A measure that is not well understood is useless and costly to the measurer and should be discarded. If a measure cannot be placed easily into any existing taxon, the taxonomy may be incomplete. An incomplete taxonomy calls for more research.

The thirty-two metrics with which the table has been populated have been validated in the narrow sense of Fenton [9] using measurement theory with Zuse's [30] augmentation [18]. Fifty measures found in the literature ([6], [7], [16], and [17]) were subjected to validation via measurement theory. Twenty of these measures were found to be valid in the narrow sense of Fenton [9]. Every measure that passed validation in the narrow sense fit unambiguously into this taxonomy. Twelve new measures have been validated and added to these [18]. An attempt was made to fill in those cells that lacked validated measures. The attempt was successful in filling in fifteen cells for which validated measures did not previously exist. Additional measures were added to five cells that may have previously had incomplete measurement.

8. CONCLUSIONS

Often there are many metrics which attempt to measure the same dimension of the same level. The collection of measurement data is very expensive [8]. However, the collection of multiple measures to measure the same dimension of the same level of software can be useful. The collection of multiple measures allows them to be compared to each other to either confirm that they do indeed measure the same dimension or establish that one (or more) of them is measuring something other than the dimension in question. Once it is established which measure most cost effectively measures the dimension in question, it may not be necessary to collect the other measures. If the measures in one cell are not all measuring the same dimension, then one or more of the measures may have been miscataloged. It is left to the measurer to determine which measures to use.

Though many of the fourteen dimensions appear multiple times in the literature, they may not be the dimensions that matter. There may be other dimensions that do not yet have metrics designed to measure them but which must be measured in order to understand an object-oriented artifact. Certainly all fourteen dimensions will not matter for all levels. Once cells are identified as being irrelevant they should be Xed out or otherwise marked as such. The same dimension measured on different levels will almost certainly require different measures or at least a different scope, e.g., lines-of-code (LOC) in a program vs. LOC in a system.

Some measures may be scaleable to levels other than that level for which they were designed. Measures that are scaleable are not directly applicable as defined but may lend themselves to being averaged or summed to fill a cell at a higher level. No measures have been found to be scaleable to cells at a lower level.

Table 2: Object-Oriented Measures Taxonomy

Level ⇨ Dimension ⇩	Variable	Method	Class	Program	System
Clarity			CLM	CLM	CLM
Cohesion	*(1/(vfi+vfo+vp))*	*(local (mfi+mfo) / total (mfi+mfo))*	**DMC** **DCWO**	**DMC** **DCWO**	**DMC** **DCWO**
Coupling	*(1-(1/(vfi+vfo+vp)))*	*(remote(mfi+mfo) / total (mfi+mfo))*	**UCGU** **DCBO**	**UCGU** **DCBO**	**UCGU** **DCBO**
Complexity, inter-structural	*remote vfi* *remote vfo*	*remote mfi* *remote mfo* *remote I/Ov*	NIM PIM RFC	AIM PIM RFC	AIM PIM RFC
Complexity, intra-structural	*local vfi* *local vfo*	**SML** *local mfi* *local mfo* *local I/Ov*	**IMC**	**IMC**	**IMC**
Complexity, psychological		**MAA** *I/Ov*	MPC WMC	MPC WMC	MPC WMC
Design			PRC NOM	PRC FOC	PRC FOC
Encapsulation			FFU	FFU	FFU
Inheritance	*vfd*	*mfd*	**PMI** **PMIS**	**DAC** NAC	**DAC** NAC
Information hiding			**PrIM**	**PrIM**	**PrIM**
Polymorphism	*vp*	*mp*	*(vp+mp)*		
Reuse	*vfi-1*	*mfi-1*	RUS CRE	RUS CRE	RUS CRE
Size		LOC AMS	LOC AMS NIV	LOC AMS AIV	LOC AMS AIV
Specialization			**POM** NCM NMA	**POM** NCM NMA	**POM** NCM NMA

Measures from [18]

Measures from [6], [7], [16], and [17]

Metrics from [24]

Measures that can be scaled up to a higher level or derived from scales at a lower level

Future research

The taxonomy needs to be tested empirically. If meaningful measures cannot be defined for a cell, e.g., encapsulation of a variable, then the cell should be expunged. Likewise, if useful outside variables (performance, schedule, or cost) cannot be defined against which to test the measures of a cell then the cell should be expunged. If all levels of a dimension have been expunged, the entire row (dimension) should be removed from the taxonomy matrix. If, for whatever reason, a new dimension becomes apparent, a new row should be added to the taxonomy matrix. The new dimension should then be populated with validated measures. The measures as well as the new dimension should be subjected to the same rigorous testing, at all levels, as has previously been defined for already existing measures of already existing dimensions. These steps need to take place iteratively until software products and processes are more clearly defined and understood.

REFERENCES

[1] Archer, Clark, and Michael Stinson, “Object-Oriented Software Measures”, *Technical Report CMU/SEI-95-TR-002*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1995.

[2] Baker, Albert L., James M. Bieman, Norman Fenton, Davis A. Gustafson, Austin Melton, and Robin Whitty, “A Philosophy of Software Measurement”, *The Journal of Systems and Software*, Vol. 12, 1990, p. 277-281.

[3] Briand, Lionel C., Khaled El Eman, and Sandro Morasca, “Theoretical and Empirical Validation of Software Product Measures”, *International Software Engineering Research Network technical report #ISERN-95-03*, 1995a.

[4] Briand, Lionel C., Khaled El Eman, and Sandro Morasca, “On the Application of Measurement Theory in Software Engineering”, *International Software Engineering Research Network technical report #ISERN-95-04*, 1995b.

[5] Briand, Lionel C., Sandro Morasca, and Victor R. Basili, “Property-Based Software Engineering Measurement”, *IEEE Transactions on Software Engineering*, Vol. 22, No. 1, 1996.

[6] Chen, J-Y, and J-F Lu, “A New Metric for Object-Oriented Design”, *Information and Software Technology*, p.232-240, 1993.

[7] Chidamber, Shyam R., and Chris F. Kemerer, “A Metric Suite for Object Oriented Design”, *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, June 1994.

[8] Deutsch, Michael S., and Ronald R. Willis, *Software Quality Engineering: A Total Technical and Management Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[9] Fenton, Norman, *Software Metrics: A Rigorous Approach*, Chapman & Hall, London, UK, 1991.

[10] Fenton, Norman, “Software Measurement: A Necessary Scientific Basis”, *IEEE Transactions on Software Engineering*, Vol. 20, No. 3, March 1994.

[11] Grady, Robert B., and Deborah L. Caswell, *Software Metrics: Establishing A Company-Wide Program*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.

[12] Henderson-Sellers, B., *A Book of Object-Oriented Knowledge*, Prentice Hall, NY, 1992.

[13] Hitz, Martin, and Behzad Montazeri, “Chidamber and Kemerer’s Metric Suite: A Measurement Theory Perspective”, *IEEE Transactions on Software Engineering*, Vol. 22, No. 4, April 1996.

[14] Hong, Sa Neung, Michael V. Mannino, and Betsy Greenberg, “Measurement Theoretic Representation of Large, Diverse Model Bases”, *Decision Support Systems*, 10, 1993.

[15] Kyburg, Henry E., Jr., *Theory and Measurement*, Cambridge University Press, Cambridge, UK, 1984.

[16] Li, Wei, and Sallie Henry, "Object-Oriented Metrics that Predict Maintainability", *Journal of Systems and Software*, Vol 23, p.111-122, 1993.

[17] Lorenz, Mark, and Jeff Kidd, *Object-Oriented Software Metrics*, Prentice Hall, Englewood Cliffs, NJ, 1994.

[18] Neal, R.D., *The Validation of Proposed Object-Oriented Software Metrics By Measurement Theory*, Virginia Commonwealth University, Dissertation.

[19] Roberts, Fred S., *Measurement Theory with Applications to Decisionmaking, Utility, and the Social Sciences*, Addison-Wesley Publishing Company, Reading Massachusetts, 1979.

[20] Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, NJ, 1991.

[21] Schneidewind, Norman F., "Methodology for Validating Software Metrics", *IEEE Transactions on Software Engineering*, Vol. 18, No. 5, May 1992a.

[22] Smith, David N., *Concepts of Object-Oriented Programming*, McGraw-Hill, NY, 1991.

[23] Sully, Phil, *Modeling the World with Objects*, Prentice Hall, NY, 1993.

[24] Tegarden, David P., Steven D. Sheetz, and David E. Monarchi, "A Software Complexity Model of Object-Oriented Systems", *Decision Support Systems 13*, p. 241-62, 1995.

[25] Tukey, John, "Data Analysis and Behavioral Science or Learning to Bear the Quantitative Man's Burden by Shunning Badmandments", *The Collected Works of John W. Tukey, Vol.III*,Wadsworth, 1986.

[26] Vollman, Thomas E., "Software Quality Assessment and Standards", *Computer*, June 1993.

[27] Weyuker, Elaine J., "Evaluating Software Complexity Measures", *IEEE Transactions on Software Engineering*, Vol. 14, No. 9, September 1988.

[28] Zuse, Horst, *Software Complexity: Measures and Methods*, Walter de Gruyter, Berlin, 1990.

[29] Zuse, Horst, "Foundations of Validation, Prediction, and Software Measures", *Annual Oregon Workshop on Software Metrics*, April 20-22, 1994.

[30] Zuse, Horst, "Foundations of Object-Oriented Software Measures", *Proceedings of the Third International Software Metrics Symposium*, March 1996.

Ralph D. (Butch) Neal is a research associate at the National Aeronautics and Space Administration (NASA)/ West Virginia University (WVU) Independent Verification and Validation (IV&V) Facility in Fairmont West Virginia. Butch has over 30 years experience in data processing mostly from a business perspective. He was president of Augusta Computer Systems (a small software development company) before returning to school. He has a BA and an MBA from WVU and a Ph.D. from Virginia Commonwealth University. Research interests include software measurement (s/m) for reuse, s/m for quality control, and s/m of rapid software development processes.